Reflektion

När en order når *Kitchen* är den ett objekt bestående av 3 nycklar: *menus, order-ID* och *status* (figur 1). Status berättar var i tillberedningsprocessen orden ligger och order-ID:t är helt enkelt kundens ordernummer. Menus värdepar är en array bestående av objekt som representerar en burgare. Dessa har nycklarna *ingredients* (array med ingredienser), *price* (burgarens pris), *units* (antalet burgare som beställts av denna variant) och *itemCount* (fyller ingen funktion i Kitchen men innehåller en array som räknar antalet ingredienser från varje produkt-kategori).

En order visas först i *Grill View* under kolumnen *Inkomna* för att därefter flyttas till nästa kolumn (i det här fallet *Tillagas*) när någon trycker på den gröna pilen (figur 2). Klickar en på det röda krysset tas ordern bort. En miss i vår implementering är att det inte dyker upp någon slags varning vid klick på den röda knappen och inget sätt att ångra borttagningen av en order, detta kan ha förödande konsekvenser i köket om detta görs av misstag samt att det lär användaren att det är farligt att testa knappar. Vidare finns en kritisk bugg vid borttagning av order: ordernumret minskar med 1 (kod 1). Om två ordrar ligger inne, #1 och #2, och order #1 tas bort kommer nästkommande order också få ordernummer #2. Det finns inte heller någon implementation av en tillbaka-knapp, det vill säga att order-statusen ändras till föregående värde. Detta gör att ett feltryck på den gröna knappen kan förstöra arbetsflödet i köket. Vidare har hänsyn tagits till färgblinda kockar då knapparna är, utöver färgkodade, även markerade med unika symboler. Detta hade dock kunnat göras ännu bättre, tillexempel genom att ha olika luminans på de röda och gröna knapparna för att särskilja dessa ännu tydligare från varandra.

Presentationen av en order är starkt influerad av så kallade ”bongar”, det vill säga papperslappar med orderinformation som används i de flesta kök, detta för att förkorta inlärningstiden för användaren så mycket som möjligt. Unikt för ordrar under kolumnen Inkomna är att enbart ingredienser från produkt-kategorin protein visas (kod 2 & 3), detta för att övriga ingredienser inte antas vara relevanta för kockarna i detta skede. I det övre högra hörnet visas dessutom totala antalet ”pattys”, d.v.s. köttpuckar eller proteinet, en order innehåller för att ge kockarna en snabb överblick över hur mycket plats som krävs på stekbordet (kod 4 & 5). En miss här är att kod 4 är onödigt komplicerad, just nu loopar vi över funktionen calculateOccurences (kod 5) filtrerat på protein och summerar värdeparet för alla nycklar. En mer effektiv lösning hade varit att använda sig av en for-loop och hämta arrayen som kopplas till nyckeln itemCount för varje meny i en order, hämta värdet från det array-index som innehåller räknaren för produkt-kategorin protein och multiplicera detta med värdeparet för nyckeln units. På så vis behövs enbart 2 värden vid varje iteration istället för att loopa över alla ingredienser för alla burgare. Ett annat eventuellt problem är att allt som är protein inte nödvändigtvis behöver stekas, friterad halloumi till exempel läggs antagligen i fritösen och inte på stekbordet, vilket är något vi i dagsläget inte tar hänsyn till eller har en snabbfix för. En lösning är att skapa ännu fler kategorityper och ha flera räknare för tillexempel stekbord och fritös, istället för bara stekbord som det är nu.

För att ta hänsyn till att alla kockar jobbar olika finns det möjlighet att byta hur ingredienserna presenteras, detta sker genom att trycka på knappen *byt vy*. Skillnaden är att den presentation som visas i figur 2 slår ihop alla ingredienser från alla burgare och visar det totala antalet, medan vyn i figur 3 separerar ingredienserna efter burgare och sorterar ingredienserna efter produkt-kategori. Arrayerna som de olika vyerna baseras på sparas dock inte, utan räknas om för alla ordrar varje gång som knappen ”byt vy” klickas på. Antagligen är antalet ingredienser för varje order för litet för att detta ska skapa problem, men i värsta fall kan detta orsaka frysningar på grund av beräkningstiden om många långa ordrar ligger under kolumnen Inkomna. En lösning är att skapa två nya arrayer, alternativt objekt, direkt när en order kommer in som sparar väsentlig information för de båda vyerna och på så vis bara göra uträkningarna en gång. Detta skulle dock kosta desto mera minne istället.

Det finns även möjlighet att dölja en order för att spara plats på skärmen (figur 4), order-ID och antalet köttpuckar syns då fortfarande för att kocken med enkelhet ska se om en order kan påbörjas utan att hela tiden behöva byta mellan kompakt och utfällt läge. Ett problem med detta är att när en order expanderas och överlappar två eller fler rader trycks de understående beställningarna undan på ett sätt som kan vara förvirrande för användaren (figur 5 & 6). En lösning vore att implementera en *masonry-layout* (figur 7) för att på så vis behålla funktionaliteten i att kunna

minimera/maximera en order samtidigt som ordrarna behåller sin position och inte hoppar runt. Vidare ändrar denna knapp färg vid tryckning, från blå i kompakt läge till svart i utfällt dito. Den blåa knappen syns mycket väl, medan den svarta blir nästan osynlig (jämför order #1 och #2 mot order #4 och #8 i figur 5). En lösning vore att knappen inte byter färg alls, utan istället bara byter symbol från ”*+*” till ”*-*”.

Ett problem som genomsyrar hela Kitchen är att applikationen är anpassad efter en touch-skärm eller möjligtvis mus-anpassad, det finns inga kortkommandon för tangentbord. Kombinera mörk bakgrund, många skärmtryckningar och flottiga kock-fingrar och resultatet blir efter ett tag en skärm som knappt går att se på grund av allt fett/all smuts som fastnar. Implementering av kortkommandon från ett tangentbord vore därför ett bra tillägg.

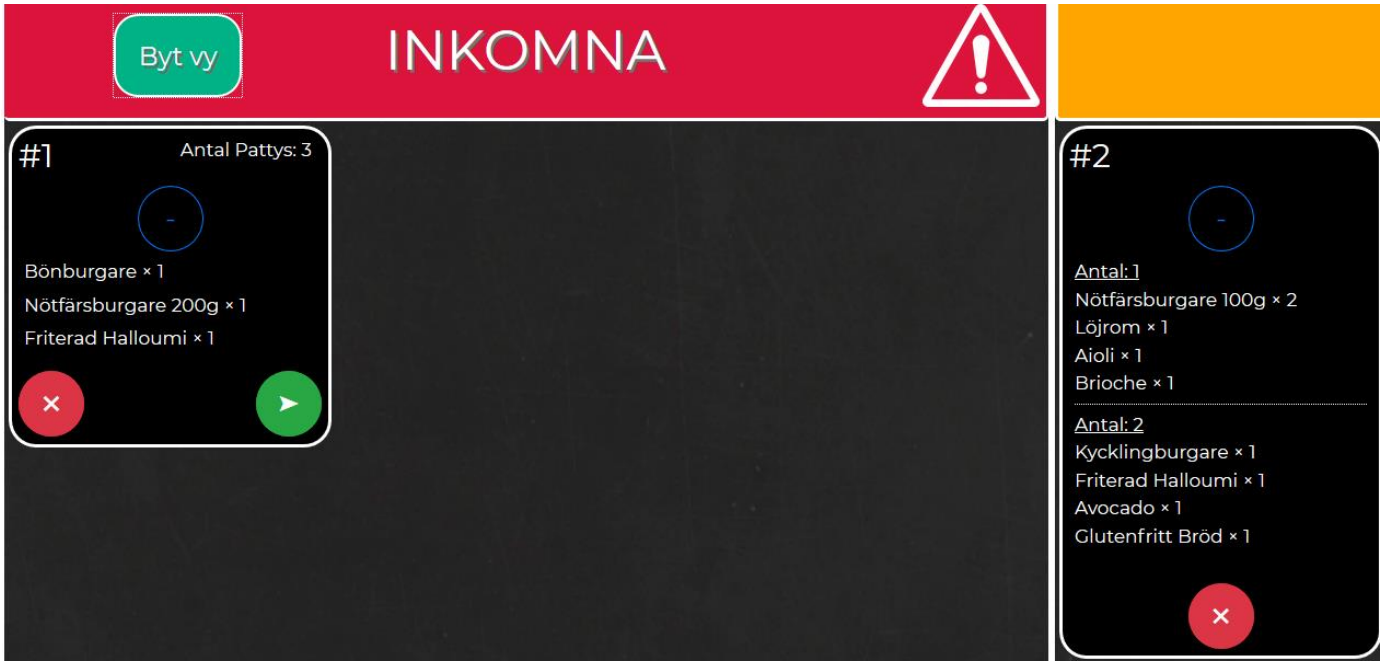
En annan aspekt vi inte tagit hänsyn till är att det inte går att lämna meddelanden till köket. En troende muslim eller äggallergiker kan tillexempel inte meddela köket att inte steka köttet på samma yta som bacon eller ägg stekts på, något som annars är vanligt förekommande i många kök.

Appendix

Figurer

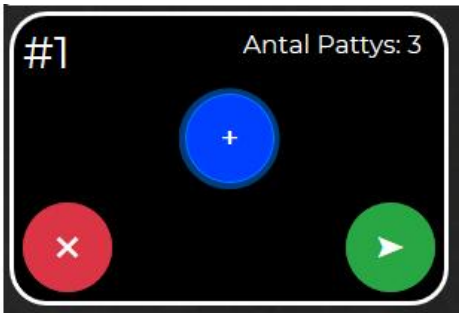

```
{ menus:
   [ { ingredients: [Array], price: 95, units: 1, itemCount: [Array] },
     { ingredients: [Array], price: 75, units: 1, itemCount: [Array] },
     { ingredients: [Array], price: 75, units: 1, itemCount: [Array] } ],
  orderId: 1,
  status: 'not-started' }
```


Figur 1. Utseendet på en order som nått Kitchen.

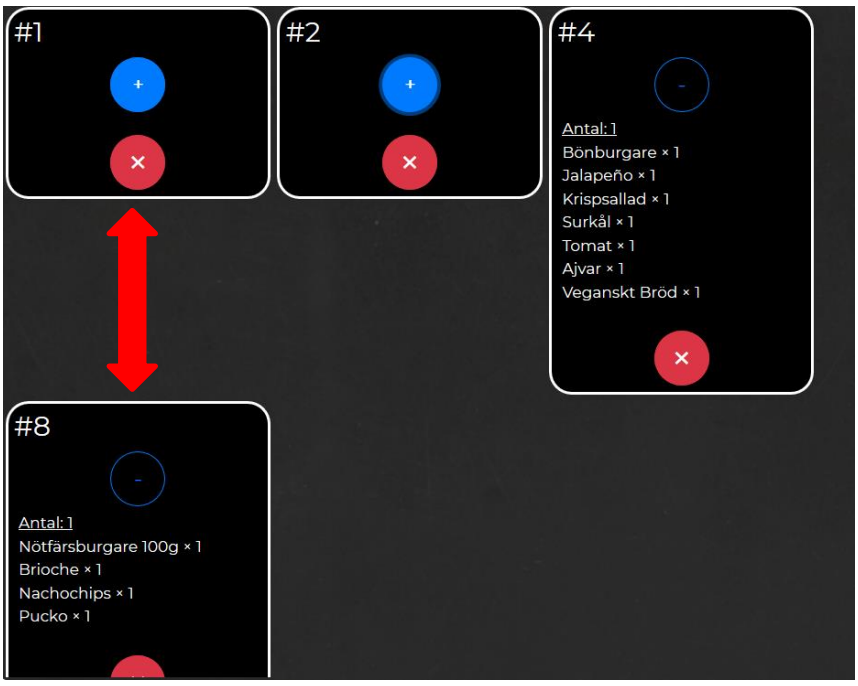


Figur 2. I Grill View, den del av Kitchen-vyn som används av kockarna som står för stekningen, representeras inkomna ordrar på följande vis. Order-ID:t står i ordrerrutornas övre vänstra hörn. Här har order #1 status not-started och ligger under Inkomna, medan order #2 har status order-started och ligger i kolumnen Tillagas. Notera att order #1 enbart visar köttpuckar, medan order #2 även visar övriga ingredienser.

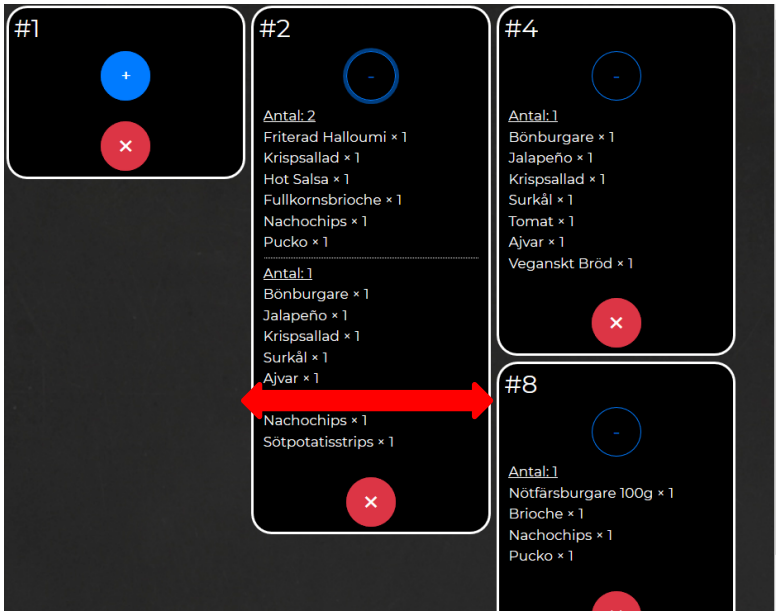
Figur 3. Alternativ vy för burgare i kolumnen Inkomna.



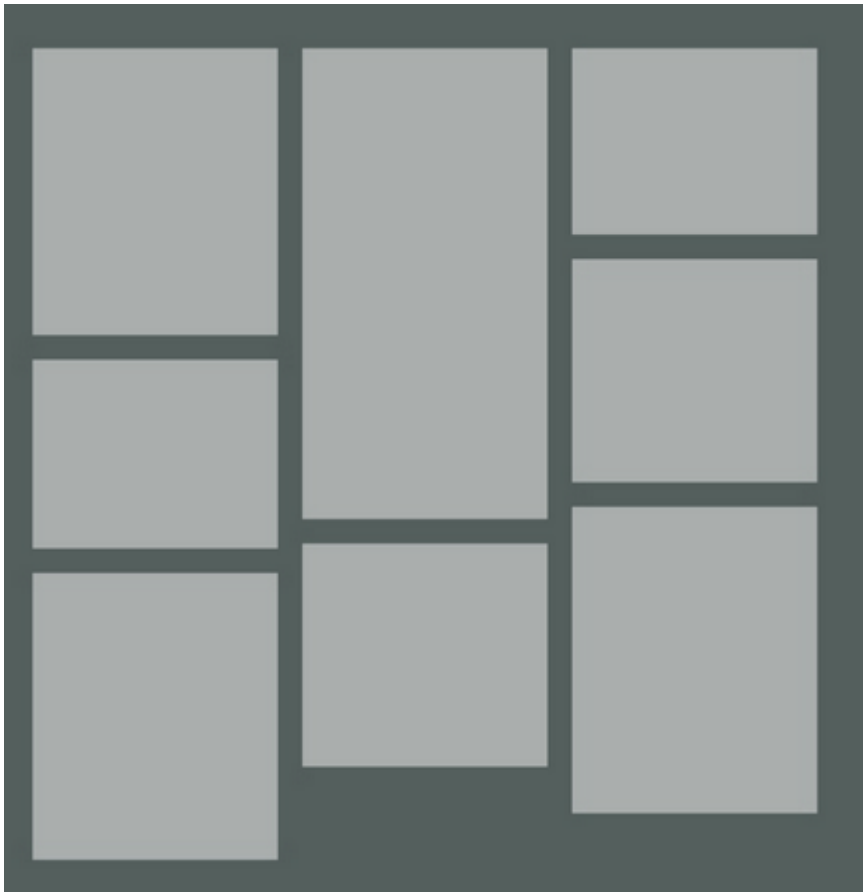
Figur 4. Minimerad order.



Figur 5. Exempel på hur ordrar på rad 2 trycks nedåt av utfällning av ordrar på rad 1. Den röda pilen visar hur order #8 förskjuts när order #4 fälls ut.



Figur 6. Vidare exempel på hur ordrar ”hoppar”, jämför med figur 5. Den röda pilen visar hur order #8 flyttas när order #2 fälls ut, trots att det finns gott om utrymme under order #1.



Figur 7. Exempel på en masonry-layout. Illustrerar en mer optimal lösning för utfällda och kompakta ordrar.

Kod

```
01.  Data.prototype.markOrderCanceled = function (orderId) {
02.    this.orders[orderId].status = 'cancel';
03.    this.currentOrderNumber -= 1;
04.  };
```

Kod 1. Funktion som anropas i DataHandler.js när en order avbryts. På rad 3 går det att se var buggen som orsakar dubbla ordernummer uppstår. En lösning är att helt ta bort rad 3.

```
01.  calculateOccurrencesWholeOrder:function(category){
02.                  let i,
03.                  flatArray = [],
04.                  mergedMenusArray=[];
05.
06.                  /*Nedan börjar vi med att plocka ut ingredienserna, en burgare i taget,
07.                  och räkna hur många gånger denna ingrediens förekommer i en burgare.
08.                  Detta antal måste sedan multipliceras med antalet av just denna burgare
09.                  som blivit beställd. mergedMenusArray har i slutet formen
10.                  {{id.1:antal.1, id.2:antal.2}, {id.a:antal.a, id.b:antal.b}*/
11.                  for(i=0; i<this.menusArray.length; i++){
12.                      mergedMenusArray.push(this.menusArray[i].ingredients.reduce((occ,ingredient)=>{
13.                          if(category!=undefined){
14.
15.                              if(ingredient.category===category){
16.                                  occ[ingredient.ingredient_id] = (occ[ingredient.ingredient_id] || 0) + 1*this.menusArray[i].units;
17.                              }
18.                          }
19.                          else{
20.                              occ[ingredient.ingredient_id] = (occ[ingredient.ingredient_id] || 0) + 1*this.menusArray[i].units;
21.                          }
22.                          return occ;
23.                      },{})
24.                  );
25.              }
26.              /*När ovanstående kod utförts kvarstår problemet att olika objekt kan
27.              innehålla samma id. Detta löses genom en ny reduce.
28.              Då varje currentValue är ett objekt med nycklar, loopar vi här över
29.              nycklarna för varje objekt och kopierar över dessas värde till accumulator,
30.              ett till en början tomt objekt som sparar nycklarna och plussar på
31.              objektens values till sitt eget för respektive nyckel*/
32.              flatArray=mergedMenusArray.reduce((accumulator, currentValue) => {
33.                  Object.keys(currentValue).forEach((key)=>{
34.                      accumulator[key]=(accumulator[key] || 0) + currentValue[key]
35.                  });
36.                  return accumulator;
37.              }, {});
38.              return flatArray;
39.          },
```

Kod 2. Funktion för att slå ihop och räkna ut totala antalet ingredienser i en order. Djupare beskrivning står i kodens kommentarer. Inparametern category är i vårt fall 1, det vill säga kategorinumret för protein.

```
01.  <div
02.  v-if="!menuView && onlypatty"
03.  v-for="(occurrences,id,occIndex) in calculateOccurrencesWholeOrder(1)"
04.  class="mg4"
05.  :key="occIndex">
06.  {{ingredients.find((item) => item.ingredient_id == id)["ingredient_"+lang]}} x {{occurrences}}
07.  </div>
```

Kod 3. Vue och html-kod för standard-vyn av ingredienserna i en order under kolumnen Tillagas. menuView är en boolean som styr vilken vy som visas, onlypatty är en boolean som styr att bara ingredienser från protein-kategorin visas och Ingredients är en array bestående av alla ingredienser i systemet. Funktionen CalculateOccurrencesWholeOrder ses i kod 1.

```
beforeMount:function(){
        /*Denna metod tar ut alla värden (dvs förekomster) av
        ingredienserna i calculateOccurrences (filtrerad på kött)
        och adderar dessa, denna siffra emittas sedan till parent*/
        let i,
        sum=0;

        for(i=0; i<this.menusArray.length; i++){
            sum = sum + Object.values(this.calculateOccurrences(i,1)).reduce(
                (sum,curr)=>sum+curr,0) * this.menusArray[i].units;
            }
            this.$emit('total_pattys',sum);

        },
```

Kod 4. Räknar totala antalet pattys/köttpuckar i en order. Djupare beskrivning står i kodens kommentarer. Funktionen calculateOccurrences kan ses i kod 5.

```
calculateOccurrences:function(index,category){

            let ingredients = this.menusArray[index].ingredients.slice();
            if(category != undefined){
                /*Filtrerar ut så enbart den aktuella kategorin räknas på*/
                ingredients=ingredients.filter((ingredient) => ingredient.category===category);
            }
            /*Nedanstående else-sats sorterar ingrediens-arrayen på kategori genom att använda JavaScript-metoden sort()
                med en manuell anonym sorteringsfunktion*/
            else{
                ingredients.sort(function(a,b){
                    /*om a.category-b.category < 0 kommer a få ett lägre index än b, = 0 kommer a och b behålla sin
                    relation i position och >0 kommer a få ett högre index än b*/
                    return a.category - b.category;})
                }
                /*Occurences är först en tom array som sparar antalet gånger ett ingredient ID förekommer*/
                let occurrences = ingredients.reduce(function(occ,ingredient){
                    occ[ingredient.ingredient_id] = (occ[ingredient.ingredient_id] || 0) + 1;
                    return occ;
                },{})
                return occurrences;
            }
```

Kod 5. Funktion för att räkna ut antalet förekomster av en ingrediens per burgare. Djupare beskrivning står i kodens kommentarer. Funktionen tar index och category som inparameter. Index berättar vilken burgare som räknas på och category filtrerar på en eventuellt angiven kategori.